Using the Shell

&

Shell Metachracters

ITER, Bhubanewar

Text Books

Brain W. Kernighan, & Rob Pike

The Unix Programming Environment

PHI

Kay A. Robbins, & Steve Robbins

Unix^{TM} Systems Programming

Communications, concurrency, and Treads

Pearson Education

Introduction

☞ Command, External command, Internal command

✍ **`$ type commandname`** Gives the location of command. The command **`$ type`** looks up only the directories specified in the **PATH** variable.

PATH
- ☞ It is a predefined shell variable
- ☞ The sequence of directories that the shell searches to look for a command is specified in the PATH variable.
- ☞ Value to PATH variable can be set.
- ✍ Example:: **`$ echo $PATH`**

Command Line
- ☞ Command withs arguments (optional options and other parameters)
- ☞ This line will be considered complete only after the user has hit the return key.
- ☞ The complete line is then fed to the shell as its input for **interpretation** and execution.

Shell

- Interface between the user and kernel.
- It is the program that interprets the user request to run programs.
- It is both a command interpreter and a programming language.
- At its base, a shell is simply a macro processor that executes commands. The term macro processor means functionality where text and symbols are expanded to create larger expressions.
- Shells may be used interactively or non-interactively. In interactive mode, they accept input typed from the keyboard. When executing non-interactively, shells execute commands read from a file.
- **Different Shells::**
  - Bourne shell (sh)
  - Bourne again shell (bash)
  - Kourn shell (ksh)
  - C shell (csh)

✍ To know type of shell **`$ echo $0`** or **`$ echo $SHELL`**

The shell's Interpretive cycle

A special unix command is running at the terminal. It is with the user all the time and never terminate unless logout. The special command is the shell. Shell (sh or bash etc.) is an external command with a difference. It possesses its own set of internal commands.

Activities Performed by the Shell in it's Interpretive Cycle

- ☞ The shell issues the prompt ($) and waits for the user to enter a command.
- ☞ After a command is entered, the shell scans the command line for metacharacters (special) and expands abbreviations to recreate a simplified command line.
- ☞ It then passes the command line to the kernel for execution.
- ☞ The shell waits for the command to complete and normally can't do any work while the command is running. (exceptions are there)
- ☞ After the command execution complete, the prompt reappears and the shell returns to its waiting role to start next cycle. User can enter another command.

Command line structure

- ☞ A single word command: `$ who`
- ☞ Multiple commands in a single line: `$ who ; date`
- ☞ Connecting commands through pipe: `$ date ; who | wc`
- ☞ Parentheses can be used to group commands:

```
$ (date ; who ) | wc
```

- ☞ `&` tells the shell not to wait for the command to complete. The `&` is used to run a long-running command in the background while the user continue to type interactive commands: `$ long-running-command &`
- ☞ The command `sleep` waits the specified number of seconds before exiting: `$ sleep 5`
- ☞ Try:: `$ ( sleep 5 ; date )& date`
- ☞ An easy way to run a command in future;

```
$ ( sleep 300; echo No Exam)&
```

✍ **The precedence of & is higher than ; (semicolon).**

Arguments to Commands

- Arguments are words separated by blanks and tabs
- Most programs accept arguments on the command line such as `$ pr filename`
- Most programs also have options, indicated by arguments begining with a minus sign.
- The various special characters (metacharacters) interpreted by the shell such as <, > ,|, ; and & are not arguments to the programs the shell runs. They instead control how the shell runs them.
- `$ echo hello >Junk`

  is identical to
- `$ >Junk echo hello`

Shell Metacharacters

- Characters that have special properties are known as matacharacters
- The shell recognizes a number of characters as special.
- The shell interprets the characters rather than passing to the command.

Character	Meaning
>	Output redirection
> >	Output redirection (append)
<	Input redirection
*	File substitution wildcard; zero or more characters
?	File substitution wildcard; One character
[ ]	File substitution wildcard; any character between brackets
|	The pipe
;	Command sequence, Sequences of Commands
` cmd `	Command Substitution (The symbol is backquote above TAB key)
$(cmd)	Command Substitution

Shell Metacharacters

Character	Meaning
( )	Group commands, Sequences of Commands
#	Comment
$var	Expand the value of a variable
\	Prevent or escape interpretation of the next character
p_1 && p_2	AND conditional execution. Run p_1; if successful, run p_2
p_1 || p_2	OR conditional execution. Run p_1; if unsuccessful, run p_2
&	Run command in the background, Background Processes
< <	Input redirection
${ var }	Parameter substitution; Check if a variable is defined & determine a value, otherwise print other.
{ }	One use in parameter substitution, other in array. For example **$** numb=(1 2 3 4 5), **$** echo ${ numb[2] }

How to Avoid Shell Interpretation

- Escape the metacharacter with a backslash (\\). Escaping characters can be inconvenient to use when the command line contains several metacharacters that need to be escaped.
- Use single quotes (’ ’) around a string. Single quotes protect all characters except the backslash (\\ ).
- Use double quotes (" "). Double quotes protect all characters except the backslash (\\ ), dollar sign ($) and grave accent (‘).
- Single and double quotes protect each other. For example:
  - ✍ `$ echo ’Hi “Intro to Unix” Class’`

    Hi "Intro to Unix" Class
  - ✍ `$ echo “Hi ’Intro to Unix’ Class”`

    Hi ’Intro to Unix’ Class

Creating New commands

- Creating new commands out of old ones.
- New commands helps to replace a sequence of command that is to be repeated more than few times.
- It would be convient to make it into a “new” command with its own name, so we can use it like a regular command.
- **Method-1::**
  - The command '`who | wc -l`' to count number of users logged in
  - Create an ordinary file named **nu** that contains `who | wc -l`
  - Run the file **nu** in two ways for the same result

1st way

- The shell, which is a program is run with its input coming from a file **nu** instead of the terminal; `$ bash <nu`

2nd way

- The shell takes its input from a file if one is named as an argument; `$ bash nu`

- `$ bash <nu` VS `$ bash nu`

Method-2 Creating New commands

- ☞ The command '**`who | wc -l`**' to count number of users logged in
- ☞ Create an ordinary file named **nu** that contains **`who | wc -l`**
- ☞ Make **nu** exacutable **`$ chmod +x nu`**
- ☞ Create a bin directory in your own pwd **`$ mkdir bin`**
- ☞ Move the file **nu** to **bin** directory
- ☞ Set the PATH variable value as **`$ PATH=$PATH:./bin`** (No space before and after = sign)
- ☞ Type **`$ nu`** to run just like regular command

Shell file:: If a file is executable and if it contains text, then the shell assumes it to be a file of shell commands. Such a file is called a *shell file*